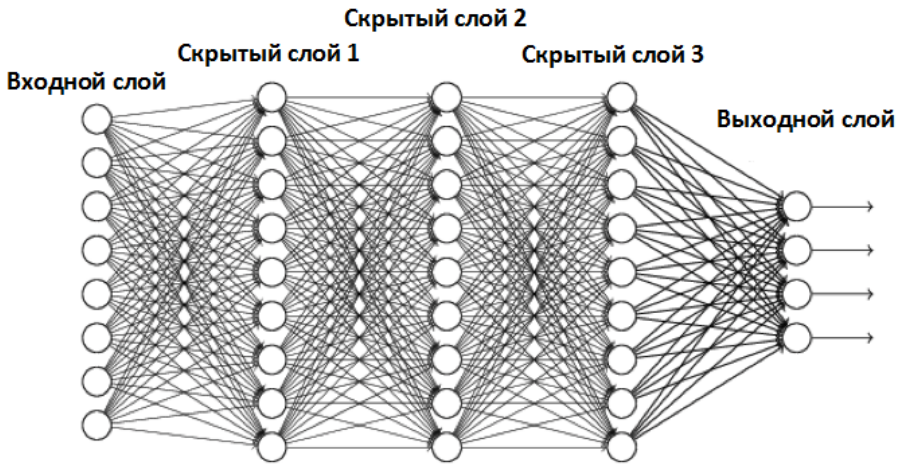

Как настроить дерево решений?

Дата публикации Nov 11, 2019

Как гиперпараметры для дерева решений влияют на вашу модель и как вы выбираете, какие из них настраивать?

Настройка гиперпараметра

> Настройка гиперпараметра - поиск в пространстве гиперпараметров набора значений, который оптимизирует архитектуру вашей модели.

Это отличается от настройки параметров вашей модели, когда вы выполняете поиск в пространстве функций, что лучше всего минимизирует функцию стоимости.

Настройка гиперпараметра также сложна в том смысле, что не существует прямого способа вычислить, как изменение значения гиперпараметра уменьшит потери вашей модели, поэтому мы обычно прибегаем к экспериментам. Это начинается с того, что мы указываем диапазон возможных значений для всех гиперпараметров. Теперь это то, где большинство застряли, какие значения я собираюсь попробовать, и чтобы ответить на этот вопрос, сначала нужно понять, что означают эти гиперпараметры и как изменение гиперпараметра повлияет на архитектуру вашей модели, тем самым попытаться понять, как ваша модель производительность может измениться.

Следующим шагом после определения диапазона значений является использование метода настройки гиперпараметра. Существует множество, наиболее распространенным и дорогим из которых является Grid Search, где другие, такие как Random Search и Bayesian Optimization, обеспечат «более умную» и менее дорогую настройку. Эти методы не являются предметом настоящей статьи, но если вы хотите узнать больше, посмотрите справочный раздел [1].

Древо решений

Дерево решений является одним из популярных и наиболее широко используемых алгоритмов машинного обучения из-за его устойчивости к шуму, устойчивости к отсутствующей информации, обработки нерелевантных, избыточных значений прогнозирующих атрибутов, низких

вычислительных затрат, интерпретируемости, быстрого выполнения и надежных предикторов. Я знаю, это много 😂. Но общий вопрос, который мне задают студенты, - как настроить дерево решений. Каков должен быть диапазон значений, которые я должен использовать для максимальной глубины, какое минимальное количество выборок требуется для конечного узла? Это очень хорошие вопросы, на которые нет однозначного ответа, но мы можем понять, как их изменение повлияет на вашу модель. Как то, что действительно означает увеличение максимальной глубины, так и изменение минимальной выборки оставляет для вашей модели. Итак, в этой статье я попытаюсь дать вам представление об этих параметрах и о том, как они влияют на архитектуру вашей модели и что она может означать для вашей модели в целом.

Давайте рассмотрим реализацию дерева решений Scikit-learn и позвольте мне объяснить, что представляет собой каждый из этих гиперпараметров и как он может повлиять на вашу модель. Кстати, обратите внимание, что я предполагаю, что у вас есть базовое понимание деревьев решений.

Поскольку дерево решений в первую очередь является моделью классификации, мы рассмотрим классификатор дерева решений.

DecisionTreeClassifier

критерий: строка, необязательно (по умолчанию = «gini»):

Функция для измерения качества раскола. Поддерживаемые критерии: «Джини» для примеси Джини и «энтропия» для получения информации.

Если вы когда-нибудь задумывались, как узлы дерева решений разделяются, то это с помощью примесей. Примесь - это мера однородности меток на узле. Существует много способов реализации измерения нечистоты, два из которых реализованы в Scikit-Learn: получение информации и Gini Impurity или индекс Gini.

Прирост информации использует меру энтропии как меру примеси и разделяет узел так, что он дает наибольшее количество прироста информации. Принимая во внимание, что Gini Impurity измеряет расхождения между распределениями вероятностей значений целевого атрибута и разделяет узел так, что он дает наименьшее количество примесей.

Согласно статье «Теоретическое сравнение между индексом Джини и критериями получения информации» [3], частота согласия / несогласия индекса Джини и получения информации составляла всего 2% от всех случаев, поэтому для всех целей и задач вы можете в значительной степени и то, и другое, но единственная разница в том, что энтропия может быть немного медленнее для вычисления, потому что она требует вычисления логарифмической функции:

$$Gini: Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

$$Entropy: H(E) = -\sum_{j=1}^{c} p_j \log p_j$$

Многие исследователи отмечают, что в большинстве случаев выбор критериев расщепления не будет иметь большого значения в производительности дерева. Каждый критерий превосходит в одних случаях и уступает в других, как предполагает теорема «Нет бесплатного обеда».

разделитель: строка, необязательно (по умолчанию = «лучший»)

> Стратегия, используемая для выбора разделения на каждом узле. Поддерживаемые стратегии являются «лучшими» для выбора лучшего разделения и «случайными» для выбора лучшего случайного разделения.

Согласно «лучшей» и «случайной» реализации scikit-learn [4], «лучший» и «случайный» разделители используют алгоритм на основе Фишера-Йейтса для вычисления перестановки массива функций Вам не нужно беспокоиться об алгоритме, единственное отличие состоит в том, что в «лучшем» разделителе он оценивает все разбиения с использованием критерия перед разделением, тогда как «случайный» разделитель использует случайную равномерную функцию с min_feature_value, max_feature_value и random_state как входы. Мы рассмотрим, что это ниже, но сейчас давайте посмотрим, как сплиттер повлияет на модель.

Допустим, у вас есть сотни функций, тогда «лучший» разделитель будет идеальным, потому что он вычислит лучшие функции для разделения на основе показателя примеси и будет использовать его для разделения узлов, тогда как если вы выберете «случайный», у вас есть большие шансы в конечном итоге с функциями, которые на самом деле не дают вам столько информации, что привело бы к более глубокому и менее точному дереву.

С другой стороны, «случайный» разделитель имеет некоторые преимущества, в частности, поскольку он выбирает набор функций случайным образом и разделяет, он не имеет вычислительных затрат на вычисление оптимального разделения. Кроме того, он также менее склонен к переобучению, потому что вы по сути не рассчитываете лучшее разделение перед каждым разделением, и дополнительная случайность поможет вам в этом, поэтому, если ваша модель переоснащается, тогда вы можете изменить разделитель на «случайный» и переобучить.

Поэтому для дерева с небольшим количеством функций без переоснащения я бы выбрал «лучший» сплиттер для обеспечения безопасности, чтобы вы получили наилучшую возможную архитектуру модели.

max_depth: int или None, необязательный (по умолчанию = None)

> Максимальная глубина дерева. Если None, то узлы расширяются до тех пор, пока все листья не станут чистыми или пока все листья не будут содержать меньше чем min_samples_split samples.

Теоретическая максимальная глубина, которую может достичь дерево решений, на единицу меньше количества обучающих выборок, но ни один алгоритм не позволит вам достичь этой точки по очевидным причинам, одной из которых является переоснащение. Обратите внимание, что это количество обучающих выборок, а не количество объектов, поскольку данные могут быть разделены на одну и ту же функцию несколько раз.

Давайте сначала поговорим о случае None по умолчанию. Если вы не укажете глубину для дерева, scikit-learn будет расширять узлы до тех пор, пока все листья не станут чистыми, то есть у листа будут только метки, если вы выберете значение по умолчанию для min_samples_leaf, где значение по умолчанию равно единице. Обратите внимание, что большинство этих гиперпараметров связаны друг с другом, и мы вскоре поговорим о min_samples_leaf. С другой стороны, если вы укажете min_samples_split, который мы рассмотрим далее, узлы будут расширяться до тех пор, пока все листья не будут содержать меньше минимального количества выборок. Scikit-learn выберет одно над другим в зависимости от того, какая из них дает максимальную глубину для вашего дерева. Здесь много движущихся частей, min_samples_split и min_samples_leaf, поэтому давайте просто возьмем max_depth изолированно и посмотрим, что происходит с вашей моделью, когда вы ее измените, поэтому после того, как мы пройдем через min_samples_split и min_samples_leaf, мы сможем лучше понять, как все это происходит все вместе.

В общем, чем глубже вы позволяете своему дереву расти, тем сложнее становится ваша модель, потому что у вас будет больше разбиений, и она будет собирать больше информации о данных, и это является одной из основных причин перенастройки в деревьях решений, потому что ваша модель будет идеально подходит для тренировочных данных и не сможет хорошо обобщать результаты тестов. Таким образом, если ваша модель переоснащается, уменьшение числа для max_depth является одним из способов борьбы с переобучением.

Также очень плохо иметь очень низкую глубину, потому что ваша модель будет недостаточно подходящей, чтобы найти лучшее значение, экспериментируйте, потому что переоснащение и подгонка очень субъективны для набора данных, и нет единого значения, подходящего для всех

решений. Итак, что я обычно делаю, так это позвольте модели сначала определить max_depth, а затем, сравнивая мои результаты в поездах и тестах, я ищу переоценку или недостаточную экипировку и в зависимости от степени, в которой я уменьшаю или увеличиваю max_depth.

min_samples_split: int, float, необязательный (по умолчанию = 2)

Минимальное количество выборок, необходимое для разделения внутреннего узла:

- *Если int, тогда рассмотрите min_samples_split как минимальное число.*
- *Если float, то min_samples_split - это дробь, а ceil (min_samples_split * n_samples) - минимальное количество выборок для каждого разбиения.*

min_samples_split и min_samples_leaf, если вы читаете их определения, это звучит так, как будто одно подразумевает другое, но нужно отметить, что лист - это внешний узел, а min_samples_split говорит о внутреннем узле, и по определению внутренний узел может иметь дальнейшее разделение, тогда как листовой узел по определению является узлом без дочерних элементов.

Допустим, вы указали min_samples_split, и в результате разбиения получается лист с 1 сэмплом, и вы указали min_samples_leaf как 2, тогда ваш min_samples_split не будет разрешен. Другими словами, min_samples_leaf всегда гарантируется независимо от значения min_samples_split

Согласно статье, Эмпирическое исследование по настройке гиперпараметров деревьев решений [5] идеальные значения min_samples_split имеют тенденцию быть между 1 и 40 для алгоритма CART, который является алгоритмом, реализованным в scikit-learn. min_samples_split используется для управления перегонкой. Более высокие значения не позволяют модели изучать отношения, которые могут быть очень специфичными для конкретной выборки, выбранной для дерева. Слишком высокие значения также могут привести к недостаточной подгонке, поэтому в зависимости от уровня недостаточной или чрезмерной подгонки вы можете настроить значения для min_samples_split.

min_samples_leaf: int, float, необязательный (по умолчанию = 1)

Минимальное количество образцов должно быть в листовом узле. Точка разделения на любой глубине будет учитываться только в том случае, если она оставляет не менее min_samples_leaf обучающих выборок в каждой из левой и правой ветвей. Это может иметь эффект сглаживания модели, особенно в регрессии.

- *Если int, тогда рассмотрите min_samples_leaf как минимальное число.*
- *Если float, то min_samples_leaf - это дробь, а ceil (min_samples_leaf * n_samples) - минимальное количество выборок для каждого узла.*

Подобно min_samples_split, min_samples_leaf также используется для управления перетеканием, определяя, что каждый лист имеет более одного элемента. Таким образом, гарантируя, что дерево не может соответствовать учебному набору данных, создавая группу небольших ветвей исключительно для одной выборки каждая. В действительности, то, что это делает на самом деле, это просто говорит дереву, что каждый лист не должен иметь примеси 0, мы рассмотрим примеси далее в min_impurity_decrease.

В статье «Эмпирическое исследование по настройке гиперпараметров деревьев решений» [5] также говорится, что идеальные значения min_samples_leaf имеют тенденцию находиться в диапазоне от 1 до 20 для алгоритма CART. В этой статье также указывается, что min_samples_split и min_samples_leaf являются наиболее ответственными за производительность конечных деревьев из анализа их относительной важности [5].

Согласно scikit-learn, мы можем использовать min_samples_split или min_samples_leaf, чтобы гарантировать, что несколько выборок сообщают о каждом решении в дереве, контролируя, какие разбиения будут учитываться. Они также говорят, что очень маленькое число обычно означает, что дерево будет соответствовать, тогда как большое число будет препятствовать тому, чтобы дерево изучило данные, и это должно иметь смысл. Я думаю, что единственным исключением является случай, когда у вас есть проблема с несбалансированным классом, потому что тогда регионы, в которых класс меньшинства будет составлять большинство, будут очень маленькими, поэтому вам следует выбрать более низкое значение.

min_weight_fraction_leaf: float, необязательно (по умолчанию = 0.)

Минимальная взвешенная доля общей суммы весов (всех входных выборок), необходимая для конечного узла. Образцы имеют одинаковый вес, когда sample_weight не указан.

min_weight_fraction_leaf - это доля входных выборок, которые должны находиться на листовом узле, где веса определяются с помощью sample_weight, это способ справиться с дисбалансом классов. Балансировка классов может быть выполнена путем выборки равного количества выборок из каждого класса или, предпочтительно, путем нормализации суммы весов выборок для каждого класса к одному и тому же значению. Также обратите внимание, что min_weight_fraction_leaf будет менее склонен к доминирующим классам, чем критерии, которые не знают весов выборки, например min_samples_leaf.

Если выборки взвешены, будет проще оптимизировать древовидную структуру, используя основанный на весе критерий предварительной обрезки, такой как min_weight_fraction_leaf, который гарантирует, что узлы листа содержат по крайней мере часть общей суммы весов выборки.

max_features: int, float, string или None, необязательно (по умолчанию = None)

Количество функций, которые следует учитывать при поиске лучшего разделения:

- *Если int, то учитывайте функции max_features при каждом разделении.*
- *Если float, то max_features - это дробь, и функции int (max_features * n_features) рассматриваются при каждом разбиении.*
- *Если «авто», то max_features = sqrt (n_features).*
- *Если «sqrt», то max_features = sqrt (n_features).*
- *Если «log2», то max_features = log2 (n_features).*
- *Если None, то max_features = n_features.*

Примечание. Поиск разбиения не останавливается до тех пор, пока не будет найден хотя бы один допустимый раздел выборок узлов, даже если для этого требуется эффективная проверка функций max_features.

Каждый раз, когда происходит разделение, ваш алгоритм просматривает ряд функций и выбирает тот, который имеет оптимальную метрику с использованием примеси Джини или энтропии, и создает две ветви в соответствии с этой функцией. В вычислительном отношении тяжело каждый раз просматривать все функции, поэтому вы можете просто проверить некоторые из них, используя различные опции max_features. Другое использование max_features - это ограничение переоснащения, выбрав уменьшенное количество функций, мы можем повысить стабильность дерева и уменьшить дисперсию и переоснащение.

Что касается выбора опций, он будет зависеть от количества имеющихся у вас функций, вычислительной интенсивности, которую вы хотите уменьшить, или от степени переоснащения, которую вы имеете, поэтому, если у вас высокие вычислительные затраты или у вас много переоснащение, вы можете попробовать с «log2» и в зависимости от того, что это производит, вы можете либо немного поднять его с помощью sqrt, либо уменьшить его, используя пользовательское значение с плавающей запятой.

random_state: int, экземпляр RandomState или None, необязательно (по умолчанию = None)

Если int, random_state - начальное число, используемое генератором случайных чисел; Если экземпляр RandomState, random_state является генератором случайных чисел; Если None, генератор случайных чисел - это экземпляр RandomState, используемый np.random.

Ха-ха пресловутый random_state, большинство новичков спрашивают меня, почему 1, почему 0 или почему 42? 42 потому что в этом смысл жизни, дух.

random_state не является гиперпараметром для настройки, или вы должны you. Позвольте мне начать с того, когда и почему вы должны установить random_state. Самый простой ответ заключается в том, что вы можете получить согласованные результаты, в некоторой степени потому, что помните разделитель, он привнесет некоторую случайность в ваши результаты, поэтому, если вы повторно запустите дерево решений, ваши результаты будут другими, но они не должны быть слишком разными.

Это подводит меня к следующему пункту: я видел, как новые студенты играют со значениями random_state и изменениями их точности, это может произойти, потому что алгоритм дерева решений основан на жадном алгоритме [6], где он повторяется несколько раз с использованием

случайного На выбор объектов (сплиттер) и на этот случайный выбор влияет генератор псевдослучайных чисел [7], который принимает значение random_state в качестве начального значения, поэтому, изменяя random_state, вы можете случайным образом выбрать хорошие функции, но вам нужно Если вы понимаете, что random_state не является гиперпараметром, то изменение точности вашей модели с помощью random_state означает, что с вашей моделью что-то не так. Это хороший намек на то, что в ваших данных много локальных минимумов, и дерево решений не очень хорошо с этим справляется, поэтому я бы предпочел, чтобы вы установили random_state и настроили другие параметры, чтобы не застревать в локальных минимумах. чем поиграть с random_state.

min_impurity_decrease: float, необязательно (по умолчанию = 0.)

> Узел будет разделен, если это разделение вызовет уменьшение примеси, большее или равное этому значению.
>
> Взвешенное уравнение уменьшения примеси имеет следующий вид:

```
N_t / N * (impurity - N_t_R / N_t * right_impurity
                    - N_t_L / N_t * left_impurity)
```

> где N - общее количество выборок, N_t - количество выборок в текущем узле, N_t_L - количество выборок в левом дочернем элементе, а N_t_R - количество выборок в правом дочернем элементе.
>
> N, N_t, N_t_R и N_t_L все ссылаются на взвешенную сумму, если sample_weight пройден.

min_impurity_decrease помогает нам контролировать, насколько глубоко растет наше дерево в зависимости от примесей. Но что это за примесь и как это влияет на наше дерево решений? Помните, в разделе критериев мы быстро рассмотрели индекс Джини и энтропию, ну, это мера нечистоты. Мера примеси определяет, насколько хорошо разделены несколько классов. Как правило, показатель загрязненности должен быть наибольшим, когда данные разделены равномерно для значений атрибутов, и должен быть равен нулю, если все данные принадлежат одному и тому же классу. Более подробное объяснение потребует от нас углубиться в теорию информации, которая выходит за рамки данной статьи, поэтому я попытаюсь объяснить, как изменение значений примесей влияет на вашу модель и как узнать, когда изменять эти значения.

Лучший способ настроить это - построить дерево решений и заглянуть в индекс Джини. Интерпретировать дерево решений должно быть довольно легко, если у вас есть знания предметной области в наборе данных, с которым вы работаете, потому что конечный узел будет иметь индекс Джини 0, потому что он чистый, то есть все примеры принадлежат одному классу. Затем вы можете посмотреть на расщепления, которые приводят к нулю индекса Джини, и посмотреть, имеет ли смысл классифицировать ваши классы как таковые или можно ли уменьшить глубину, тем самым приводя к более обобщенному дереву, если это так, вы можете увеличить min_impurity_decrease, чтобы предотвратить дальнейшее деление, потому что теперь узел не будет дальше разделяться, если примесь не уменьшится на указанное вами количество. Обратите внимание, что это повлияет на все ваше дерево, поэтому вы должны поэкспериментировать с числами, но приведенное выше объяснение должно дать вам отправную точку.

class_weight: dict, список диктов, «сбалансированный» или None, по умолчанию = None

> Веса, связанные с классами в форме {class_label: weight}. Если не дано, все классы должны иметь вес один. Для задач с несколькими выходами список dicts может быть предоставлен в том же порядке, что и столбцы y.

class_weight используется для предоставления веса или смещения для каждого выходного класса Но что это на самом деле означает: посмотрите, когда алгоритм вычисляет энтропию или примесь Джини для разделения на узле, результирующие дочерние узлы взвешиваются с помощью class_weight, давая веса дочерних выборок на основе указанной пропорции класса.

Это может быть очень полезно, когда у вас есть несбалансированный набор данных. Обычно вы можете просто начать с распределения ваших классов по весам классов, а затем, в зависимости от того, где находится ваше дерево решений, вы можете попытаться увеличить или уменьшить вес других классов, чтобы алгоритм наказывал выборки одного класса относительно другого. , Самый простой способ - указать «сбалансированный», а затем продолжить с пользовательскими весами.

Обратите внимание, что это не похоже на метод недостаточной или избыточной выборки, количество выборок в классе фактически не меняется, его вес, назначенный для него, меняется, это можно увидеть, распечатав дерево решений и значения в каждом из них. узел, и он изменится на

```
weight * (the number of samples from a class in the node) / (size of class)
```

presort: bool, необязательный (по умолчанию = False)

Стоит ли предварительно сортировать данные, чтобы ускорить поиск наилучших расщеплений при подгонке Если для параметров дерева решений по умолчанию для больших наборов данных задано значение по умолчанию, это может замедлить процесс обучения. При использовании меньшего набора данных или ограниченной глубины это может ускорить обучение.

Этот параметр довольно прост, если у вас небольшой набор данных или если вы ограничите глубину дерева, и после запуска первой итерации у вас будет несбалансированное дерево, где большинство точек данных располагаются только на небольшой части конечных узлов, используя предварительную сортировку Вам поможет.

Способ предварительной сортировки заключается в том, что он первоначально сортирует все переменные перед обучением и на каждом узле оценки использует отсортированные векторы, а после выбора лучшего разделения вы разбиваете точки данных, а также отсортированные индексы, чтобы отправить их потомку. узлы подмножества данных и подмножеств отсортированных индексов. Таким образом, вы можете применить эту идею в рекурсии. Однако обратите внимание, что это работает, если число экземпляров данных больше, чем количество входных объектов.

Резюме

Сложность дерева решений имеет решающее значение для его точности, и она явно контролируется используемыми критериями остановки и применяемым методом сокращения. Обычно сложность дерева измеряется одной из следующих метрик: общее количество узлов, общее количество листьев, глубина дерева и количество используемых атрибутов [8]. max_depth, min_samples_split и min_samples_leaf являются критериями остановки, тогда как min_weight_fraction_leaf и min_impurity_decrease являются методами сокращения.

Несмотря на то, что все это в значительной степени реализует метод остановки или сокращения, оно зависит от уровня, применяемого к модели. Если у вас есть критерий жесткой остановки, ваша модель может оказаться подгонкой, поэтому, если вы измените его на свободные критерии остановки, ваша модель может оказаться более подходящей, поэтому у нас есть методы обрезки. У нас должен быть свободный критерий остановки, а затем использовать обрезку для удаления ветвей, которые способствуют переоснащению. Но обратите внимание, что обрезка - это компромисс между точностью и обобщенностью, поэтому ваши оценки поездов могут быть ниже, но разница между показателями поездов и тестов также станет меньше.

Я надеюсь, что у вас есть лучшее представление об этих параметрах и о том, как они могут взаимодействовать друг с другом при настройке гиперпараметров. Но если что-то не понятно, пожалуйста, дайте мне знать в комментариях, и я буду более чем рад объяснить дальше.

*Не стесняйтесь добавлять меня на*LinkedIn*или следуй за мной*facebook*,*

Ссылка

[1]Тюнинг гиперпараметра для моделей машинного обучения.

[2]Scikit-Learn DecisionTreeClassifier

[3] Лора Елена Райляну и Килиан Стоффель, «Теоретическое сравнение между индексом Джини и критериями получения информацииАнналы по математике и искусственному интеллекту 41: 77–93, 2004.

[4]Внедрение Scikit-Learn Splitter

[5] Рафаэль Гомес Мантовани, Томаш Хорват, Рикардо Черри, Сильвио Барбон Младший, Хоакин Ваншорен, Андре Карлос Понсе де Леон Феррейра де Карвалью, «Эмпирическое исследование гиперпараметрической настройки деревьев решений»
Arxiv: 1812,02207

[6]Жадный алгоритм

[7]Генератор псевдослучайных чисел

[8] Лиор Рокач, Одед Маймон, «Глава 9: Дерево решений»

Оригинальная статья

- Фреймворки и библиотеки (большая подборка ссылок для разных языков программирования)
- Список бесплатных книг по машинному обучению, доступных для скачивания
- Список блогов и информационных бюллетеней по науке о данных и машинному обучению
- Список (в основном) бесплатных курсов машинного обучения, доступных в Интернете